Programarea secvențială şi programarea concurentă

Un program imperativ este compus din acțiuni şi datele asupra cărora se operează. Dacă aceste acțiuni sunt concepute de aşa natură încât se desfăşoară în mod strict secvențial, programul reprezintă un *proces* iar activitatea de programare corespunzătoare se numeşte *programare secvențială*.

Procese paralele şi procese concurente

Marea majoritate a sistemelor de operare actuale (Microsoft Windows, UNIX, OS/2) permit gestionarea multiprogramată a proceselor. Folosirea în comun de către mai multe programe aflate în memorie, a resurselor sistemului duce la o îmbunătățire spectaculoasă a gradului de utilizare al acestora.

Programele rulate sub sistemele cu multiprogramare formează prin urmare un set de *procese paralele* executate independent între ele. Comportarea lor individuală este practic identică celei din cazul rulării fiecăruia într-un sistem monoprogramat. Eventualele conflicte rezultate din utilizarea comună a unor resurse se rezolvă prin sistemul de operare, fără ca programatorul să ia vreo măsură în acest sens.

Programele care reprezintă procese izolate (programe secvențiale) nu permit rezolvarea unor anumite categorii de probleme. Deseori devine necesară prezența simultană a mai multor procese, executate în paralel şi asincron, dar între care există relații de cooperare (schimb de mesaje, transfer de date) şi care gestionează în comun resurse ale sistemului.Ele se numesc *procese concurente*.

Coexistența proceselor concurente, care în marea majoritate a timpului se desfăşoară asincron, impune în anumite momente sincronizarea în vederea realizării unei comunicări. Relațiile între ele, care fac necesar acest lucru, se pot reduce în final la următoarele două situații tipice:

a) **Excluderea mutuală** – între procese care folosesc în comun aceeaşi resursă, la care accesul este permis doar dintr-un singur proces la un moment dat. În acest caz este necesară secvențializarea cererilor de acces la resursa respectivă (partajată). Alteori se impune *sincronizarea pe condiție* a proceselor: amânarea activării unui proces până când o anumită condiție devine adevărată.

O resursă care poate fi utilizată doar dintr-un singur proces la un moment dat se numeşte *resursă critică*. Zona de instrucțiuni dintr-un proces prin care se manipulează o resursă critică reprezintă o *secțiune critică*. Cu ajutorul acestor noțiuni putem defini excluderea mutuală ca fiind acea formă de sincronizare impusă proceselor concurente care permite ca numai un proces să se afle în interiorul unei secțiuni critice la un moment dat. Este vorba desigur de secțiuni critice relative la aceeaşi resursă critică.

O primă încercare de elaborare a unei construcții unitare şi structurate pentru rezolvarea excluderii mutuale o reprezintă *regiunea critică* introdusă de C.A.R.Hoare şi P.Brinch Hansen în 1972. Regiunile critice extinse astfel încât să rezolve şi sincronizarea pe condiție se numesc **regiuni critice condiționale**.

b) **Coperarea** – între procese care schimbă mesaje sau în general date, cel mai des în relația: producător/consumator. Informațiile produse de un proces sunt utilizate de celălalt.

Spre deosebire de *programarea secvențială*, care permite definirea unui singur proces izolat, descrierea unor procese concurente şi a relațiilor dintre ele se face prin aşa-numita *programare concurentă*. Coexistența în cadrul aceluiaşi program a mai multor procese necesită protejarea datelor şi resurselor proprii proceselor față de orice acces exterior. Resursele partajabile între anumite procese vor fi de asemenea protejate față de accesul din procese neautorizate. deseori în programele concurente intervine şi factorul timp, procesele desfăşurându-se în *timp real*.

Categoria cu totul nouă de probleme apărută odată cu ivirea necesității scrierii pe scară industrială a programelor concurente a dus la cristalizarea unor mijloace tot mai simple şi generale, în vederea rezolvării lor în mod unitar. Această tendință s-a concretizat ulterior prin apariția familiei limbajelor de programare concurentă. Ele îmbină metodele programării concurente cu claritatea şi simplitatea proprie limbajelor de nivel înalt. Verificările efectuate la compilare cu privire la respectarea regulilor de acces la resursele partajate şi corectitudinea unor operații de sincronizare, evitarea blocărilor de sistem etc. permit detectarea rapidă a unor erori grave, altfel greu de găsit.

Datorită răspândirii INTERNET-ului dar şi rețelelor în general, cele mai utilizate *sisteme* concurente sunt, în prezent, cele *distribuite*. Particularitățile programării pe sisteme distribuite a impus desigur dezvoltarea unor limbaje care alături de concurență prezintă facilități noi, specifice caracterului distribuit al programelor. De exemplu, în ceea ce priveşte cooperarea, aceste sisteme comunică prin *transmitere de mesaje*.

Limbaje pentru programarea concurentă

Dintre acestea, am ales pentru exemplificare un limbaj simplu şi de dimensiuni reduse, EDISON, definit de P.Brinch Hansen în 1980. Un program Edison descrie procese concurente care comunică prin intermediul unor variabile comune. Sincronizarea proceselor se realizează cu ajutorul regiunilor critice condiționale, în maniera:

`when` *b_1* `do` *listă_de_instr_1* `else` *b_2* `do`
listă_de_instr_2 `else ...` *b_n* `do` *listă_de_instr_n* `end`

Observăm că nu se specifică variabila comună la care se referă regiunea critică. Aceasta din motivul că în Edison s-a adoptat următoarea soluție simplificatoare: se exclud reciproc toate regiunile critice. Cu alte cuvinte, la un moment dat, se execută o singură secvență critică. Rezultă astfel o simplificare importantă la implementarea limbajului, dar cu prețul unor constrângeri suplimentare în ceea ce proveşte concurența proceselor.

Instrucțiunea `when` se execută în două faze:

- faza de sincronizare: procesul este întârziat până când nici un alt proces nu execută faza critică a unei instrucțiuni `when`;
- faza critică: se evaluează pe rând expresiile logice (condițiile) *b_1*, *b_2*, `...` . Dacă se găseşte o condiție adevărată, se execută secvența de instrucțiuni corespunzătoare, după care se încheie instrucțiunea `when`. Dacă toate condițiile sunt false, se revine la faza de sincronizare.

Descrierea activităților concurente se face cu ajutorul instrucțiunii `cobegin`:

`cobegin` constr_1 `do` listă_de_instr_1 `also` constr_2 `do` listă_de_instr_2 `...` const_n `do` listă_de_instr_n `end`

Listele de instrucțiuni reprezintă procesele care ce se vor executa în paralel. Ele iau naştere în momentul în momentul executării unui `cobegin` şi dispar după ce s-au încheiat toate, moment în care s-a terminat instrucțiunea `cobegin`. Fiecărui proces i se atribuie cîte o constantă; sensul asociat ei nu este precizat în limbaj şi poate fi diferit de la o implementare a acestuia la alta (necesarul de memorie pentru procesul respectiv, numărul procesorului asociat, prioritate etc.).

Un program Edison are forma unei proceduri. El se lansează în execuție prin activarea corpului de instrucțiuni al acestei proceduri. Programul este alcătuit, de obicei, din mai multe module. Identificatorii exportați (descrişi în modul dar vizibili şi din exteriorul lui) sunt marcați prin caracterul * în fața declarației.

Unele dintre limbajele concurente se adresează cu precădere unui anumit domeniu cum ar fi **programarea în timp real** sau, mai general, *programarea sistemelor încorporate* (*embedded systems*). Aplicațiile în timp real cer ca sistemul să răspundă la un eveniment exterior, într-un interval de timp finit. Este necesar deci ca programul să aibă acces la un ceas de timp real, fie pentru a asigura întârzierea unui proces pe o anumită perioadă de timp fie pentru a declanşa un eveniment (proces) la o anumită oră.

Scurt istoric al dezvoltării limbajelor de programare

Primele limbaje de programare de nivel înalt au apărut încă din anii 50 .

Primul a fost FORTRAN proiectat de un grup de la IBM condus de John Bachus (1954). O influență majoră în ceea ce priveşte dezvoltarea ulterioară a limbajelor de programare (chiar dacă a fost relativ puțin utilizat practic) l-a avut limbajul ALGOL 60 (1958-1960); contribuția majoră a acestuia au fost *structura pe blocuri şi procedurile recursive*. Ca urmare a unui efort finanțat de Departamentul Apărării al S.U.A. (DOD), în 1959 a fost definit limbajul COBOL, destinat în special aplicațiilor economice şi de gestiune; au apărut pentru prima dată *fişierele* şi *facilitățile pentru descrierea datelor* (precursoare ale înregistrărilor de astăzi (`record, struct`). Este interesant de remarcat că până în ziua de astăzi FORTRAN şi COBOL (chiar dacă nu în forma lor inițială) sunt încă mult utilizate.

În aceeaşi perioadă, sfârşitul anilor 50 începutul 60, au apărut şi primii reprezentanți ai celorlalte familii de limbaje : LISP (McCarthy, MIT-1960) şi APL (Iverson, IBM-1962), primele limbaje funcționale, şi SNOBOL (Bell Laboratories, 1964), primul limbaj cu caracter declarativ.

Astfel că până la mijlocul anilor 60 circula deja o diversitate considerabilă de limbaje de programare. În această perioadă, IBM a lansat un proiect ambițios, de a defini un limbaj care să reunească toate conceptele cunoscute, urmând să devină o sinteză a acestora şi să înlocuiască toate limbajele existente. Astfel în 1964 a apărut PL/I, care s-a “bucurat” însă de un succes foarte redus, fiind deosebit de complex şi greoi.

În continuare anii 60 au adus încă două limbaje de importanță majoră. ALGOL 68 (1968) a adus o serie de îmbunătățiri predecesorului ALGOL 60. Limbajul excelează prin grija cu care s-a urmărit asigurarea unei *ortogonalități* perfecte. Limbajul este definit cu multă acuratețe, inclusiv prin metode formale. Importanța lui este în special teoretică, exercitând o influență majoră asupra dezvoltării în continuare a limbajelor de programare. SIMULA 67 (1967) este de asemenea un urmaş al limbajului ALGOL 60, fiind prevăzut cu facilități speciale pentru simulare. Importanța lui constă însă în introducerea conceptului de *clasă* ca prim mijloc de modularizare şi descriere a unor date abstracte.

În 1971 N.Wirth a definit, pornind de la ALGOL 60, un limbaj de mare succes – Pascal. Cheia succesului l-a reprezentat expresivitatea deosebită prin opoziție cu simplitatea sa.

În 1974 a apărut C, unul dintre cele mai răspândite limbaje în momentul de față (Dennis Richie, Bell Labs -1974).

Anii 70 au adus o serie de cercetări în vederea elaborării unor metode avansate de programare şi a implementării lor în limbaje : *tipuri de date abstracte*, *verificarea programelor*, *tratarea excepțiilor*, *programarea concurentă*. Au rezultat limbaje ca Mesa (Terax, 1974), Concurrent Pascal (Hansen, 1975), CLU (Liskov, MIT-1974). De o mai largă utilizare se bucură în prezent Modula 2 (Wirth, 1977) şi, în special, Ada (DOD, 1979). Şi Ada a însemnat o încercare de sinteză de gen PL/I, cu mult mai izbutită desigur, dar care deocamdată nu s-a impus în măsura în care s-a sperat la inițierea proiectului.

Tot în anii 70 a apărut PROLOG (Colmerauer, 1972), cel mai de seamă reprezentant al limbajelor logice, deosebit de actual astăzi în special datorită aplicațiilor în domeniul inteligenței artificiale.

Anii ’80 au adus o proliferare a cercetărilor în domeniul limbajelor funcționale, unde alături de LISP (foarte mult utilizat şi dezvoltat) au apărut limbaje noi cu inovații importante: SML (Milner, Edinburgh – 1984), Miranda (Turner, Kent – 1985), Haskell (Hudak – 1988) etc. Au continuat de asemenea cercetările în jurul limbajului Prolog (în special adaptarea la arhitecturi moderne neconvenționale. În domeniul limbajelor imperative s-a remarcat influența modelului de proiectare/programare orientat pe obiecte. Primul limbaj orientat pe obiecte (de fapt un întreg mediu de programare) a fost Smalltalk, realizat de firma Xerox la sfârşitul anilor 70. În 1988 a apărut C++ (Straustrup, Bell Labs), o dezvoltare a limbajului C în direcția programării orientate pe obiecte, foarte mult folosit în prezent. Proiecte moderne şi puternice sunt Object Oberon (Zürich 1989) şi Eiffel (B. Meiyer 1988) şi, în mod abstract, Java (1995, Sun Microsystems Inc.). Java este un limbaj de programare orientat pe obiecte, cu facilități deosebite pentru interactivitate şi animație, destinat pentru programarea aplicațiilor pentru INTERNET şi, în general, a celor distribuite.

Spre deosebire de C şi C++, din care este inspirat, s-a renunțat la anumite facilități considerate nesigure: aritmetica pointerilor, eliberarea explicită de memorie, moştenirea multiplă.

La succesul deosebit pe care îl are, în prezent, limbajul JAVA, Microsoft (MS) a răspuns cu un proiect propriu: limbajul de programare C#.

Lansarea versiunii alfa a avut loc în anul 2000, şi a fost realizată de către un colectiv de la MS condus de ANDRES HEJLSBERG.

La fel ca şi Java, C# este şi el un limbaj derivat din C şi C++.